CptS 515 Advanced Algorithms HOMEWORK-1

Sharath Kumar Karnati

September 2023

---

1. In Lesson 1, algorithm's complexity is measured on input size instead of input values. Please indicate the input size for an algorithm that solves the following problem:

- **GIVEN:** A number n and two primes p,q
- **QUESTION :** Is it the case that n = p · q?

Solution: For the above given problem, To determine n= p.q, we can do multiplication on p and q and then we can see whether the output is equal to n.

Now,
The total size that it takes for the multiplication will be:

- I. To store p, it takes $\log_2(p)$.
  II. To store q, it takes $\log_2(q)$.
  III. To store n, it takes $\log_2(n)$.
- Now,the total space will be : size of p + size of q + size of n.
  Total Space = $\log_2(p)$ + $\log_2(q)$ + $\log_2(n)$
  Total Space = $\log_2(pqn)$

  Therefore, the total space required for the above algorithm will be $\log_2(pqn)$

2.In Lesson 2, we learned linear-time selection algorithm where the input array of numbers are cut into groups of size 5. Show that, when the group size is 7, the algorithm still runs in linear time.

Solution:

We have used a recursive algorithm to find the ith smallest element of an array of n numbers.

We have previously solved when n=5, now for n=7 we have to do the following:

- Firstly, cut the array into n/7 groups.
- Now, each group will have 7 members.
- Then, sort each group.
- Now we will have n/7 medians for n/7 groups after sorting each group.

- Recursively, select the n/14th smallest(Median of medians) from the n/7 medians by using a linear time selection algorithm.
- Now, swap the Median of Medians(MM) with the first element in the original array of n numbers.
- Run partition on the resulting array of numbers n, so that the MM will be placed at "r".(O(n)time)
- Then, if (i == r), return MM (ith element), (or)
  if (i > r), run recursively to select the ith smallest from the "LOW"(subarray with indices less than r). (or)
  if (i < r), run recursively to select the (i - r)th smallest element from "HIGH"(subarray with indices greater than r).
- Now, there are a total of n/7 medians for each group of size 7 we also have n/14 of medians that are exactly less than MM.
- There will also be 4 members in each of these groups that are less than or equal to the group's median.
- Therefore, we will have at least 4n/14 elements in the original array that are less than or equal to MM.
- Similarly, we will have at least 4n/14 elements greater than or equal to MM.
- Now,
  Size of LOW $\geq$4n/14,
  Size of HIGH $\geq$ 4n/14 ;
  Also, $|LOW| + |HIGH| = n$;
  $Max\{|LOW|, |HIGH|\} \leq$ 10n/14;

Worst Case Time Complexity:
Tw(n)= Tw(n/7) + Tw(Max { $|LOW|, |HIGH|$}) + $O(n)$
$Tw(n) = Tw(n/7) + Tw(10n/14) + O(n)$.
Lets assume that Tw(n) = O(n) ; as Tw(n) < c.n .
Proof:
$Tw(n) = Tw(n/7) + Tw(10n/14) + a.n$
$Tw(n) \leq$ c(n/7) + c (10n/14) + a.n
Tw(n) $\leq$ 12n/14. c.n + a.n
Tw(n) $\leq$ c.n for very large c (where c>>a)

Hence, it's proved that even after changing the group size the algorithm still runs in linear time.

---

3. In Lesson 2, we learned closest pair algorithm that runs in O(n log n) time. However, that algorithm can be improved further when additional assumption is made. Here is one. Suppose that there are n2 bugs sitting on a piece of paper of size n by n. Any two bugs must stay away by at least 1. Each bug is given as a pair of coordinates. Design a linear-time algorithm that finds the closest pair of bugs. (Hint: since the input size is n^2, the linear time here really means the running time is O(n^2) where the n^2 is the number of bugs.)

Solution:

Goal: To design a linear time algorithm. First, we must find how many bugs are there in each square of size 1×1 on a plane. Then these square can be further divided into 4 sub-squares of 1/2 × 1/2 each. So, there can be only 1 bug in each of these sub-squares as the given condition states that the distance between each bug should be at least 1. So, the total number of bugs in the 1×1 square will be "4".

If a square of size 1×1 is empty, then there must be some other square of same size with more than one bug in it. This will be true according to Dirichlet's drawer principle, as there are n^2 bugs so there must be n^2 1×1 squares available to occupy into them. We should store the coordinates of all the bugs into a matrix and then we should do local search to find the closest pair.

Now let us consider a circle of radius 2 inside which these boxes are equipped. Therefore, the number of boxes in this area will be $(\pi\ r^2)/l^2$
r= radius of the circle
l= length of the side of a square.
Area = (3.14 × 4) / 1.
Area =12.56 ≈ 13.
So, the number of boxes that are present in the area of circle will be 13.
Now, to find the closest pair of bugs we have to do :
Max number of bugs in a box × number of boxes × total bugs.
$= 4 \times 13 \times n^2$
$= 52n^2$

We know that,
$O(n^2) = C.n^2$
Here C=52,
Therefore, we can state that the above algorithm can be run in a linear time of $O(n^2)$.